Jouons avec l'authentification Windows

Une attaque via FireWire

Mini-tutoriel rédigé par Mathuin .

Tags : **décompilation** **patching** **reversing** **FireWire** **Windows** **sécurité**

Télécharger (JSON)

- Article

Article

FireWire et DMA

Le port FireWire, aussi connu sous le doux nom d'IEEE1394 appartient à une famille d'interfaces ayant un accès direct à la mémoire physique, sans la moindre restriction, et ce, pour des raisons de performance. Bien sûr, ce mécanisme n'est pas là par hasard, c'est pour éviter qu'un périphérique très rapide soit bridé par un processeur qui est à la traîne. C'est également le cas des ports Thunderbolt, PCMCIA, PCI par exemple.

Sous Windows, la mémoire est partagée entre deux *rings*: le *ring* 0 et le *ring* 3. Le *ring* 0 appartient au noyau, le *ring* 3 à l'espace utilisateur. Lorsqu'on veut accéder à une page en mémoire, un *flag* (appelé U/S) est testé: si le processus appelant est accrédité pour accéder à cette page, c'est-à-dire qu'il est de niveau de *ring* (CPL) inférieur ou égal à celui nécessaire. Sinon, on obtient une erreur de segmentation. Ce système de protection, appelé *paging*, évite qu'on fasse n'importe quoi avec le noyau, comme par exemple installer un *rootkit*, lancer du code malveillant, ou encore récupérer des identifiants. La liste des possibilités est infinie.

Or, le port FireWire n'est pas soumis au *paging* : un périphérique FireWire peut aller modifier des pages mémoires appartenant au noyau. C'est ce que fait `inception`[1] : on relie deux ordinateurs entre eux via un câble, et ensuite, on peut siphonner le contenu de la RAM ou la modifier. Une modification intéressante permet de pouvoir se connecter avec n'importe quel mot de passe, en éditant la routine de vérification du mot de passe.

Aperçu du processus d'authentification: LSASS

Au démarrage, une fois le noyau chargé en mémoire, ainsi que les services lancés, `winlogon.exe` est lancé. Ce dernier démarre `logonui.exe`, qui se charge de collecter les informations de connexion: identifiant et mot de passe, ensuite de quoi `lsass.exe` (Local Security Authority) se chargera de les vérifier. Si le mot de passe est valide, la session s'ouvre. On veut pouvoir se connecter avec **n'importe quel** mot de passe, il faut donc altérer `lsass.exe`. LSASS recourt à une DLL particulière pour la vérification des identifiants, c'est `msv1_0.dll`, notre cible désignée.

Ouvrons donc cette DLL avec IDA Pro, logiciel de désassemblage très puissant. On peut constater que Microsoft a la gentillesse de nous fournir les symboles de débogage. En clair, on nous fournit les noms des variables et des routines tels qu'elles sont nommés dans le code source.
En parcourant la liste des routines, on tombe sur la bien nommée routine `MsvpPasswordValidate`, non exportée. Désassemblons-la aussi sec. Un petit bout de code se révèle particulièrement intéressant :

```
1 validation:
2     mov esi, 10h
3     lea rdx, [rbp+50h]  ; Source2
4     mov rcx, rbx     ; Source1
5     mov r8, rsi      ; Longueur
6     call    cs:__imp_RtlCompareMemory
7     cmp rax, rsi
8     jnz mauvais mot de passe    ; Sauter si Source1 != Source2
```

La DLL provient d'une installation de Windows 64bits; c'est donc normal de retrouver des registres comme `rax`. La fonction `RtlCompareMemory` prend en entrée deux blocs de mémoire et une longueur. On compare les blocs, octet par octet, jusqu'à n octets (valeur passée en dernier paramètre). La fonction renvoie le nombre d'octets identiques (le dernier paramètre si les blocs de mémoires sont identiques).
La convention d'appel x64 est la suivante : premier paramètre dans `rdx`, puis `rcx`, `r8`, `r9`, et le reste dans la pile. On comprend alors ce que fait ce bout de code : d'abord, on met l'adresse des deux blocs de mémoire dans les registres qui vont bien (lignes 3 et 4), puis on met le nombre de bits à comparer, qui était dans le registre `rsi`, dans `r8` (ligne 5). Enfin, on lance notre appel à `RtlCompareMemory` (ligne 6).
Lors du retour de la routine, la valeur renvoyée est mise dans `rax`. En comparant `rax` et `rsi`, on compare le nombre d'octets identiques avec le nombre d'octets à comparer (ligne 24).

Si `rax` est égal à `rsi`, le *flag* `ZF` (zéro) est activé. Enfin, à la ligne d'en-dessous, on saute à l'instruction située à l'adresse référencée par l'étiquette `mauvais_mot_de_passe` si le *flag* `ZF` n'est pas activé, c'est-à-dire si `rax` est différent de `rsi`. Notons que la comparaison ne s'effectue pas entre des mots de passe en clair, mais entre le mot de passe haché contenu dans la base SAM, et le mot de passe saisi par l'utilisateur, lui aussi haché.

On veut rendre tout mot de passe valide. En désactivant le saut, on continue comme si le mot de passe avait été validé. Ce n'est dès lors même pas la peine de laisser l'appel et la comparaison en place. On va tout *nopper* : on va remplacer chaque octet correspondant à l'instruction (sous architectures x64 et x86, les instructions n'ont pas toutes la même taille) par la valeur hexadécimale 90, qui est l'instruction NOP: elle ne fait absolument rien.

C'est ce que fait `inception` : on cherche une signature en mémoire correspondant au bout de code à désactiver, on remplace les instructions par NOP et voilà, le tour est joué.

Conclusion

Cette attaque *via* port FireWire permet donc de modifier à la volée la mémoire, sans aucune restriction. Outre le déverrouillage de session, un usage beaucoup plus intéressant serait d'implanter un *rootkit* sur un serveur lorsqu'on a un accès physique. Pour se protéger, on pourrait désactiver les *drivers* IEEE1394 si on n'en n'a pas besoin.

Le *patch* de `msv1_0.dll` est également utilisé par le *script* de Metasploit `screen_unlock.rb`, qui déverrouille une session à distance, ou encore par le CD Kon-Boot.

---

1. http://www.breaknenter.org/projects/inception/ ↩

© Progdupeupl 2013 - 2014 v1.1-184-g05697c8